Mobile Web Services for Peer-to-Peer Applications

Guido Gehlen, Linh Pham
RWTH Aachen University
Communication Networks
Kopernikusstr. 16, 52074 Aachen
{guge, lph}@comnets.rwth-aachen.de

Abstract— The role of the Service Oriented Architecture (SOA) is going to be more and more popular in the software design for distributed systems. This middleware framework tries to simplify as much as possible the software architecture of distributed applications. In infrastructure based networks like the Internet the SOA is successfully implemented by XML Web Services in a client-server manner. In ad-hoc systems without any infrastructure, the Peer-to-Peer (P2P) paradigm is attractive. Thus, a Web Service realization using the P2P paradigm is promising, since a uniform middleware can be achieved without any constraints of the underlying network and communication paradigm.

In this article, we look at a heterogeneous environment being composed of mobile nodes with computing and communication capabilities. Each (peer) node equally acts as both, server and client. These nodes provide their services to other nodes in the distributed environment, and they are able to use remote services. We distinguish and discuss different SOA realizations enabling P2P computing and introduce a common software architecture for P2P Web Services. On this basis, we introduce an implementation of P2P Web Services on Java enabled mobile devices. We focus on the server implementation, enabling mobile devices (phones and PDAs) to provide and publish their services.

I. INTRODUCTION

With the expansion of mobile communication into IP based networks, new data oriented services will be possible. So far, the degree of connectivity of mobile devices is low. Mobile services are merely restricted to e-mail, instant messaging, or accessing web pages. The amount and frequency of data transmission is low.

Two trends are observable. First, wireless technologies such as GSM/GPRS, UMTS, WLAN, and BlueTooth are making bandwidth more and more inexpensive. Wireless bandwidth is continuously increasing as well as the mobile Internet subscriber [1]. The second trend is the increase of processing power and device capabilities. Mobile and handheld devices are often more powerful than personal computers of the ’90s. They provide features like digital photography, speech recognition, personal information management (PIM), and gaming.

Thus, the challenge is to build mobile distributed applications, maximizing the productivity of the whole distributed system. Not only the user has to interact with his devices (human-to-machine interface), also the devices among each other have to cooperate (machine-to-machine interface). To cope with the challenge of designing software on distributed systems with multitude of machine-to-machine interfaces, the SOA is a promising concept.

The SOA has been successfully applied within the Internet by the so called Web Services. But Web Services rely on a fixed infrastructure and a reliable network. Ad-hoc systems, especially mobile ad-hoc networks (MANETs), cannot provide a fixed infrastructure. MANETs are characterized by permanently arriving and leaving nodes in the environment. So, services are very unreliable. If a basic service of the SOA, like the broker, is unreachable, the whole environment is broken. Thus, the traditional server-centric computing is not applicable in ad-hoc systems.

In contrast, a P2P system with equal nodes does not imply a fixed infrastructure. Also, from the ubiquitous computing point of view, a P2P system has some advantages. In the future, each equipment has computing and communication capabilities and offers the environment its special services. In the same manner these devices want to consume services from other devices to work efficiently.

In one following section of this article we introduce the P2P provision of Web Services within a mobile environment. First of all, we give a short introduction to SOA and Web Services.

II. SERVICES ORIENTED ARCHITECTURE

Object oriented software architecture is hierarchically structured to build complex and reusable software. On the lowest level, functionalities are encapsulated in an object. A set of interacting software objects is collected into a component. The Service Oriented Architecture (SOA) [2] consequently extends this hierarchical structure to distributed systems and defines Services as a collection of components. Services have the following characteristics [3]:

- Services are self contained and modular
- Services are discoverable and dynamically bound
- Services stress interoperability
- Services are loosely coupled, reduction of artificial dependencies to their minimum
- Services have a network-addressable interface
- Services have coarse-grained interfaces in comparison to finer-grained interfaces of software components and objects
- Services are composable

These characteristics should make the distributed system as simple as possible, but no simpler1. In other words, the

1Motivated by Albert Einsteins’ famous statement “Things should be made as simple as possible, but no simpler”
SOA is an architectural style to achieve loose coupling among interacting software agents, i.e. to minimize their artificial dependencies.

The SOA defines three roles, a Service-Requestor (R), Service-Provider (P), and a Service-Broker (B). A software agent interacting with other software agents plays one or more roles. They communicate in the way as depicted in figure 1.

![Service Oriented Architecture](image)

Providers publish their services to a service registry (service-broker). More than one service-broker within the environment have to replicate their service registries (dashed arrow). Requestors use the Broker to search for services and integrate them by accessing the service description (dashed-dotted arrow). This description includes all information needed to access the service and is used to generate a proxy object. The proxy object represents the remote service, i.e. all published remote service methods are methods of the local proxy object. This architecture bridges the native messaging inside the client environment to the platform independent messaging in the SOA environment, see also section V-A and figure 6. The process of proxy object generation maps the platform independent description into a real software object for the client’s system environment. In our implementation a WSDL to J2ME generator is used to create Java proxy classes for a Java2 Micro Edition (J2ME) environment, see section V-A.

To achieve high interoperability, all SOA entities have to use a common language for service description, messaging and service registration. The Extensible Markup Language (XML) is such an appropriate common language. On the basis of XML the World Wide Web Consortium (W3C) specifies a middleware following the SOA. For messaging the Simple Object Access Protocol (SOAP) [4], [5] is used. It is based on standard internet protocols like HTTP or other arbitrary protocols like WAP [6] or Block Extensible Exchange Protocol (BEEP). The SOAP envelope is structured in XML. Interfaces are described in an XML subset, the so called Web Service Description Language (WSDL) [7]. This description includes all the information needed in order to invoke service methods from other nodes.

Web Services can be published by service providers and located by service requestors by means of the Universal Discovery Description and Integration (UDDI) [8], [9]. It is a directory model, the so-called yellow pages of Web Services, whose entries consist of three parts. The “white pages” contain information about the Web Service provider, the ”yellow pages” include industrial categories based on standard taxonomies and the ”green pages” describe the interface of the service, e.g. by integrating the WSDL file.

The SOA realization of Internet Web Services acts on a server-centric paradigm, as depicted in figure 2. Client agents use services from server agents. The roles are clearly separated among the nodes of the distributed system. Nodes are depicted as squares in figure 2. Each web service domain is managed by only one service-broker. The service requestor (R) nodes search and integrate services using the broker node (dashed-dotted arrows). The broker can be implemented as an UDDI server. Service providers publish their services to the broker node (dotted arrows).

![SOA - Server Client Realization](image)

This architecture is reasonable in high reliable fixed networks, since the communication amount to the broker is less compared to a decentralized architecture. Also in mobile infrastructure based systems with internet access, the so called nomadic systems, where mobile nodes only act as service requestors, the client-service architecture is reasonable. But if some service providers are mobile nodes, services will be unreliable, since the mobile communication is unreliable. Thus, the whole middleware will be unreliable.

### III. Peer-to-Peer Mobile Web Services

P2P computing is not clearly defined, each working group has its own definition.

Conceptually, P2P computing is an alternative to a centralized client-server model, but there is no clear border between them. A P2P environment has no fixed assignment of client and server nodes. All peer nodes are equal concerning their role to act as both server and client. One famous P2P application is the exchange of files (e.g. music or video files) with P2P systems like Napster or Gnutella. Each node has a public storage including the files to publish. Other nodes can access to this storage and download the files.
An important aspect is the way of sharing the content. There are three general classes of algorithms, a centralized directory model, a flooded request model, and a document routing model [10]. The first algorithm requires some managed infrastructure, since each peer has to connect to a central directory. The second requires no infrastructure, it is a pure P2P model. Each request is flooded (broadcasted) to directly connected peers, which themselves flood this request. The third algorithm uses hash values of the shared documents to route towards the peers.

We want to generalize this particular P2P file sharing application to P2P Web Services within the SOA. This means that each node publishes its services as Web Services enabling other nodes to access and to use these services. This seems simple. However, in a real mobile environment, composed of infrastructure based and ad-hoc networks, there are several difficulties:

- How and where should a service be published and discovered (centralized or decentralized by flooding or document index routing)?
- The peer nodes (device) must be addressable, but in most of current mobile networks as GSM/GPRS the IP address is covert behind a Network Address Translation (NAT)
- Today’s mobile devices have limited processing power and less memory that cause difficulties by providing server functionalities
- The mobile communication system is unreliable and offers no continuous Quality of Service (QoS).

The service register/discovery problem depends strongly on the underlying network architecture. Thus, we propose to place the responsibility for register/discovery to a lower communication layer, which has information about the network state and features. All nodes at the middleware layer should use a virtual service-broker instance, which is responsible to do the register/discovery task in a well adopted manner. For example, an environment without any service-brokers could realize this using the flooded request model, like in Universal Plug and Play (UPnP) [11], or by routing the services with a hash of the service description.

The problem of addressing each node is not the focus of this article, it could be solved with Mobile IP Version 6 (MIPv6) or P2P overlay networks, like Juxtapose (JXTA) [12].

Mainly, we will meet the challenge of the last two problems regarding restrictions of a mobile environment. The P2P service environment for mobile devices we propose enables application developers to publish arbitrary services as XML Web Services and easily integrate remote Web Services in local applications. This leads to an intelligent environment with nodes talking to each other in a common language (XML). Each node makes its services available so that the productivity of the service environment is increased to a maximum.

A. P2P Service Oriented Architectures

We distinguish between two different P2P SOA realizations. The first one, depicted in figure 3, stays on top of an infrastructure based network like the mobile Internet (WLAN, GPRS). There we have a fixed infrastructure (DHCP, DNS and UDDI server), thus, a unique service-broker. In the figure 3 all P2P nodes act as client and server. The broker is a stand-alone node.

![Fig. 3. P2P SOA Realization with fixed infrastructure](image)

Each node registers its services to the broker (dotted arrows) and searches for remote services (dashed-dotted arrows). The denotation of the symbols and arrows in the figures is deducted from figure 1. The phase of service registering, searching, and integration is done in a centralized client-server manner, since we can use a fixed infrastructure. The use of services will be in P2P manner (continuous arrows).

A second P2P web service scenario is depicted in figure 4. It will be used in a pure ad-hoc network without any infrastructure. All the time, mobile nodes are arriving and leaving the service environment. No centralized service-broker is available, instead, some nodes include a service-broker. This heterogeneous environment is characterized by nodes with a big range of capabilities and computation power. Some devices are only consumers (requestors) of services, like e.g. remote controls, terminals, tablet PC’s. Other devices are only service providers, like e.g. light controller, media server. Some devices are both service providers and requestors, thus, they publish their own services and consume remote services.

We consider, at least one node within the environment has to provide broker capabilities enabling the service exchange and adaptation of the service environment. Networks without any broker have to use flooding or index routing algorithms. This architecture as depicted in 4 copes with the matter of fact that, in a real environment, heterogenous nodes reside with highly differentiated capabilities. In addition, the Mobile Ad-hoc Network (MANET) is very dynamic, so, the communication traffic volume for managing services with e.g. a flooding algorithm will be enormous.

Thus, some devices which are preferably less dynamic and more powerful take the broker role. More than one broker within the environment must synchronize the service repository to ensure a consistency (dashed arrows). As mentioned, the middleware only requests one virtual broker regardless of
how many brokers are available. If there is more than one broker, the virtual broker request has to be routed to the best available broker node. One promising method is an Anycast-Routing with application-defined keys [13], [14]. In this case, we can use the key "service-broker" to assign it to the Anycast-Routing algorithm. So, the algorithm selects the best available broker node from a set of equal broker nodes.

Recapitulating, implementing the SOA on top of infrastructure based and ad-hoc networks in a common manner, e.g. using Web Service standards, an overall middleware which enables a consistent interexchange of services can be realized. Thus, the application developer is able to easily integrate remote services into his application. He does not have to think about network architectures and discovery algorithms. Using Web Service standards has the additional advantage that the mobile middleware has access to a variety of existing Internet Web Services. The protocols will be uniform, and existing development tools and processes can be efficiently utilized.

IV. APPLICATION FRAMEWORK FOR P2P MOBILE WEB SERVICES

In this section, a common mobile P2P application framework which enables P2P Web Services on mobile nodes will be discussed. These considerations are still detached from an implementation on a specific platform (hardware and software). The following section gives an introduction to a real implementation on J2ME devices, like most of the mobile phones and PDA’s.

The core software component in figure 5 is the SOAP-component which provides the Simple Object Access Protocol (SOAP) including an XML parser, transport protocols (HTTP, WAP, BEEP, ...), security algorithms, and the client/server ports. This component is the interface to the distributed environment and is responsible for message exchange, encryption/signature, and marshalling.

Client proxy objects, the representation of remote services, are generated and updated by the Remote Service Repository inside the Management component. This task can be done automatically without developer’s interaction by extracting the information from the WSDL file and building software objects on this basis. These objects will map on runtime local method invocations into remote SOAP calls.

Services, deployed within the framework, will publish their service descriptions in WSDL to the Local Service Repository component. Incoming requests from other nodes will be handled by a server component, the SOAP content will be analyzed, and the corresponding service will be invoked.

The two repositories in the Management component realize the service register/discovery mechanisms. Thus, they may take the role of a service broker or connect to a remote service broker. The QoS component is responsible for monitoring the network interfaces (available bandwidth, latency) and the load of the device.

If the device supports a Graphical User Interface (GUI), the GUI Adaptation module reconfigures the GUI depending on the client proxies which need a user input. This module can be used to access on runtime remote services with a default user interface.

Some of the functional elements of the architecture are optional, depending on the device capabilities.

V. P2P SOA IMPLEMENTATION FOR J2ME DEVICES

The previous section has outlined the general P2P software architecture. Now, an implementation for Java enabled mobile devices will be introduced. The Java environment is particularly suitable for the software development designed for different target platforms. The smallest Java configuration is the J2ME. Most of the mobile devices, like PDA’s or mobile phones maintain J2ME. Besides, a J2ME software implementation is also executable on other Java versions, like Personal Java or Java2 Standard Edition (J2SE).

The core component (SOAP) in figure 5 is implemented on the basis of an Open Source SOAP implementation named kSOAP [15]. The used XML parser, which is also an Open Source library, is kXML [16]. In addition, we have developed a SOAP transport binding to Wireless Application Protocol (WAP) [6] and Web Service security [17] components. In this article, special focus is given to the following parts: client proxy generation, SOAP server implementation and service deployment. The adaption of the GUI and the QoS component are not in focus of this article.
A. Client Proxy Generation

As mentioned in section II, a proxy object represents the remote service on the local device and allows invoking remote methods as if they appear locally. Since the WSDL description of the service contains all the information to access and use the service, it is possible to generate automatically the proxy class from the WSDL description, see figure 6.

For this purpose, we have developed a WSDL2Java proxy class generator [18]. This tool generates the proxy object from a WSDL file and all complex types defined in the WSDL as separate classes. These classes can be integrated in the mobile application and after compiling, local Java method calls are mapped to remote SOAP calls.

In future, a dynamic service integration on runtime is scheduled. With this and a dynamic GUI adaptation, a service could be directly used after accessing its service description and default GUI description.

B. SOAP Server Implementation

This section introduces the realization of a light-weight SOAP server for Java enabled mobile devices (J2ME). In addition, the performance characteristics of the light-weight SOAP server are described. A detailed description of the light-weight SOAP server and its performance analysis are given in an additional paper [19].

1) SOAP Server Core Architecture: The light-weight SOAP server has been designed and implemented in order to deploy Web Services on mobile devices with a J2ME Java Virtual Machine (JVM). Currently, the SOAP server makes services available via a Hypertext Transport Protocol (HTTP) transport interface, but also further transport bindings are imaginable.

Figure 7 shows the core structure of the light-weight SOAP server.

When a client connects to the SOAP server for sending a HTTP POST/GET request, the server socket accepts the client connection and returns a socket containing an input data stream. From this input data stream, a HTTP request message is extracted and redirected to the Request Handler (point 1 in figure 7) for further processing.

The Request Handler makes use of kSOAP and kXML to process the request message. After de-serializing XML data structures to Java objects, the Request Handler will pass these objects and necessary information (i.e: required parameters to invoke service methods like requested service method name, information about type and name of the general Java objects) to the Deployment Interface (point 2 in figure 7). Then, the Request Handler will call the Response Handler to handle a response to the client (point 3 in figure 7).

The Response Handler gets a result from the Deployment Interface (point 4 in figure 7) and sends the result to the client through an output data stream corresponding to the client connection socket. Depending on the type of result (a responded SOAP message or a static resource such as a file), the Response Handler will invoke the relevant method.

In order to serve client requests, service methods that were initiated at the start-up phase of the SOAP server (point 0 in figure 7), communicate with the Deployment Interface.

The Deployment Interface contains a static instance of a Hashable class. This hashtable is unique (for memory saving purpose) and stores mapping-pairs between a service method name and a service method object which is deployed. A service method must register its name and an instance of its class with the Deployment Interface once at the start-up phase of the SOAP server. Thus, a Method Name - Service Object pair is added to the hashtable.

After receiving the general Java object and the requested service Method Name from the Request Handler (point 2 in figure 7), the Deployment Interface then searches the Hash-table for the requested Method Name and invokes a corresponding service object.

2) Concurrent request handling: Under limitation of the J2ME Application Programming Interface (API), 2 strategies used to handle concurrent threads (1 thread handles 1 client request) are implemented. These strategies are also used to limit a maximum number of running threads.

With the first strategy, the algorithm is summarized as following:

1. Start generating \( n \) threads to handle \( n \) client requests in parallel.
2. Wait until the last thread (the \( n^{th} \) thread) finishes. No thread is created during this interval.
3. Start again generating \( n \) other threads and repeat the step 2.

The condition for \( n \) such that total number of running
threads will not exceed a threshold will be given in an additional paper [20] [19].

This strategy has its pros and cons. The pros is that it is very simple to implement. In addition, there are only few times that the SOAP server has to handle the client requests in parallel. Most of the time, the SOAP server only processes the client requests one by one. Therefore, this strategy is also an option for handling concurrent requests. The cons is that device’s resources are not used efficiently because the device has to wait for all of n threads to be finished. Then, the device starts generating n other threads.

The second strategy is an implementation of a thread pool. The thread pool is given some fixed number of threads (an array of \( n_p \) threads, for instance) to use. The thread pool will assign a task (processing a client request) to each of these \( n_p \) threads. When any of these threads finish its old task, a new task is assigned immediately. If the number of concurrent client requests is larger than \( n_p \), extra client requests are put in a queue for later processing by any free threads.

In comparison with the first strategy, this strategy uses device’s resources more efficiently because it allows a fixed number of threads to handle client requests continually. However, since the thread pool creates an array of \( n_p \) Thread objects to pool the tasks, more memory is allocated for this array. For example, to maintain the waiting state for client requests, the SOAP server with the Thread Pool mechanism consumes approximately 125 KB memory with 5 threads whereas the first strategy requires about 105 KB of memory no matter how many threads in one group will be created at a time.

3) Memory footprint: For the version running on J2ME CDC/PersonalProfile (and also JVM of J2SE platform), the light-weight SOAP server core package has a size of 22 KB. It is the size of the cnsoap.server package. The other packages like cnsoap.util, cnsoap.service can be variant in sizes. It depends on how many util-classes and service-classes are contained in the packages.

For run-time memory test, the J-Sprint tool is used. It is a shareware Java profiler which does performance analysis of Java applications\(^4\). Heap memory size of current J2ME mobile phones can vary. For example, Siemens S55 has the heap size of 350 KB whereas the more powerful Nokia 6600 has 3 MB of heap size.

The peak of memory usage of the lightweight SOAP server is approximately 295 KB and happens at the initialized phase when Java classes and dependent libraries are firstly called. When entering a stable working phase, about 125 KB of memory is needed to maintain the waiting state of the SOAP server. An additional memory amount of 45 KB is required by the SOAP server to handle each client request (echo service tests).

4) Serving time: A round-trip delay test between the lightweight SOAP server and the Apache SOAP running on the Tomcat servlet engine was done. An iPaq HP 3870 with Familiar Pocket Linux was used in the test.

Two groups were tested. The first group used the light-weight SOAP server and J2ME CDC/Personal Java Profile. The second group used Apache SOAP 2.3.1 and Tomcat 3.2.4 running on Blackdown Java 1.3.1 ARM version (installation packages were considered to be suitable for hardware resource of the iPaq).

The interop test was composed of 11 echo service tests (echoString, echoInteger, echoFloat, etc.). The test result stated that the round-trip delay time produced by the first group was approximately 50 percent of that produced by the second group [20] [19].

C. Service Deployment

Figure 8 shows a class diagram of the echoStruct service method which echoes an object of the Data type.

\(^4\)Information available at http://www.j-sprint.com
java.lang.Long, java.lang.String, java.lang.Boolean), that correspond to SOAP elements containing no sub-element, can be serialized and de-serialized automatically.

VI. CONCLUSIONS

In this paper we propose the Service Oriented Architecture (SOA) and the Web Service realization as a promising concept to simplify application development for mobile distributed P2P systems. We have introduced and discussed different P2P Service Oriented Architectures in general.

A P2P software architecture which has been mapped to a real Java environment for mobile devices has been designed. This environment is capable of using and providing XML Web Services. In detail, we have explained the generation of proxy objects, the SOAP server component for mobile devices and the service deployment on the mobile SOAP server. Eventually, this server component has been analyzed in terms of response time and memory usage.

In conclusion, this Mobile Web Service architecture is a promising possibility enabling mobile devices to collaborate in a P2P manner within a distributed environment. The use of XML Web Services as middleware framework stress interoperability in a heterogenous device environment. The middleware is platform and communication system independent and fits into the concept of the Model Driven Architecture (MDA) defined by the Object Management Group (OMG) [21].

ACKNOWLEDGMENT

The work presented has been motivated by a cooperation with Ericsson Research Germany within the german BMBF project INVENT-VMTL3. The authors would like to thank Prof. B. Walke and Dr. G. Gebhardt of ComNets for their help and support, as well as T. Dinsing and M. Gerdes of Ericsson Research.

REFERENCES


3Information available at http://www.invent-online.de